EXPERT COUPLING IN MOE PRETRAINING: REDUCING ALL-TO-ALL OVERHEAD WITH CORRELATED PLACEMENT AND TOKEN SHUFFLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) layers replace the feed-forward block of a Transformer with E expert networks, and each token is routed to k of these experts. Under expert parallelism (EP) the experts are distributed across GPUs, and every MoE layer runs all-to-all collectives in the forward and backward passes to dispatch tokens to their experts and then combine the results. On nodes of eight MI300X GPUs, these collectives can take 45 % of the training step at EP32 with top-2 routing and 60 % with top-6 routing. We find that early in pretraining routers have already learned to assign tokens to experts in correlated patterns, both within a layer and across layers. At top-2, 0.8 % of the expert pairs in a layer are selected together by 45 % of tokens, and the experts a token selects at one layer predict the experts it selects at the next layer. We use these correlations to keep more token–expert assignments on the token's own GPU, which reduces communication across GPUs and across nodes. Correlated expert placement puts experts that are often selected together on the same GPU. Combined with a dispatcher that sends each token to each GPU once, it removes up to 58 % of dispatched rows. Token shuffling applies when sequence parallelism shards tokens across the EP group. It moves each token to the GPU predicted to hold its next-layer experts during the reduce-scatter that follows attention. On one node this raises the share of token–expert assignments served on the token's GPU from 12.5 % to 59 %. In Megatron-LM, across EP degrees from 8 to 64 with top-2 and top-6 routing, the two methods reduce all-to-all time by 1.16–2.63× and end-to-end step time by up to 1.41×. Neither method changes the models' underlying routing decisions or expert parameters.

1 INTRODUCTION

Mixture-of-Experts (MoE) layers have become a central building block of large language models (Jiang et al., 2024; Liu et al., 2024; Yang et al., 2025; Kimi Team, 2026). In an MoE layer, the single feed-forward network (FFN) of a dense Transformer block is replaced by a set of E parallel FFNs, called *experts*, together with a lightweight router (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). After the attention sub-layer, the router scores each token's hidden state against all experts and selects its top-k experts. The token is processed only by those experts, and their outputs are combined using the router weights. Model capacity therefore grows with E, while the FLOPs per token grow only with k. Because each expert is updated only by the tokens routed to it, experts become specialized over the course of pretraining (Dai et al., 2024; Xue et al., 2024).

This capacity comes at a communication cost. Under expert parallelism (EP) (Lepikhin et al., 2021), experts are sharded across GPUs, and every MoE layer requires two all-to-all (A2A) collectives. The *dispatch* sends each token from the GPU that owns it to the GPUs holding its selected experts, and the *combine* returns the expert outputs to the owning GPU. Both collectives run in the forward pass and again in the backward pass. The communication volume grows with the EP degree and with top-k, and once the EP group spans multiple nodes, the slower inter-node links dominate step time. In our measurements on nodes of eight MI300X GPUs (Section 2.1), all-to-all communication accounts for 13.5 % and 44.8 % of step time at EP8 and EP32 under top-2 routing, and for 24.0 % and 60.4 % under top-6 routing.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Conventional MoE implementations place experts contiguously across the EP group, with expert e on GPU $\lfloor e/(E/\mathrm{EP}) \rfloor$, irrespective of which experts are used together. Depending on the routing, the all-to-all can also become unbalanced, and the few GPUs that receive a disproportionate share of tokens become stragglers for the whole collective. Mechanisms that address this include expert replication (Liu et al., 2024) (Zhai et al., 2023) and dynamically planned redundant experts with weight prefetching (Chen et al., 2026; Kimi Team, 2026). These techniques lower the *peak* per-link traffic by spreading it across GPUs. The *total* traffic stays about the same, because the same fraction of token–expert assignments crosses a GPU boundary. Replication also costs memory for the expert copies. DeepSeek-V3 additionally reduces inter-node traffic with device-limited routing, which restricts the experts the router can select.

A second line of work exploits expert *affinity*. ExFlow (Yao et al., 2024) uses inter-layer expert affinity to co-locate experts for MoE inference. This approach does not transfer directly to pretraining. During autoregressive decoding, a token can be placed on whichever GPU is convenient. During pretraining, a token stays on the GPU that holds its part of the sequence for the attention sublayer, and moving it requires additional communication. Exploiting affinity during training therefore requires a mechanism that ExFlow does not provide.

Our starting point is that once experts specialize during pretraining, routing follows stable patterns. We study expert routing across pretraining stages and load-balancing methods, and find two such patterns. The first is *within-layer correlation*. Some pairs of experts are selected together much more often than independent routing would predict. At the end of training with top-2 routing, 45 % of tokens select one of only 0.8 % of the possible expert pairs, and this pattern is already present after about one billion tokens. The second is *cross-layer correlation*. The experts a token selects at layer ℓ predict the experts it selects at layer $\ell + 1$. Both correlations appear with all three load-balancing methods we test, with different strength.

We exploit these correlations to raise *token–expert locality*, the fraction of (token, selected expert) pairs whose expert is on the token's GPU. These pairs never enter the network, so higher locality reduces the total A2A volume. Our methods require no architectural changes or expert replicas, and they do not change routing decisions or the training loss. They can be combined with the load-balancing methods above. Correlated expert placement and deduplication also apply to inference, which we do not evaluate.

We make the following contributions.

- **Correlated Expert Placement and a Deduplicating Dispatcher (Section 3.1).** We place correlated experts on the same GPU, so that the experts a token selects are more likely to share a GPU. The deduplicating dispatcher sends each token to a destination GPU at most once, however many of its selected experts are there. Deduplication is an existing technique (Zhao et al., 2025). The all-to-all dispatcher in Megatron-LM (Shoeybi et al., 2019) sends one copy of a token per selected expert, so we implement deduplication there as the base for placement. Together, placement and deduplication reduce both intra-node and inter-node communication volume.
- **Token Shuffling (Section 3.2).** For sequence-parallel layouts in which the TP group equals the EP group, we (i) predict a token's experts at layer $\ell + 1$ from its experts at the previous layers, and (ii) move the token ahead of time to the GPU that will hold those experts. The move is folded into the reduce-scatter that sequence parallelism (Korthikanti et al., 2023) already performs after attention. This brings to pretraining the locality benefit that prior affinity-based work obtained for inference.
- **Characterization of Expert Routing (Sections 2.2 and 4).** We quantify within-layer co-selection and cross-layer predictability across pretraining stages, top-k settings and load-balancing methods, and relate these statistics to communication volume.
- **Evaluation in Megatron-LM (Section 4).** We report communication volume, A2A time and end-to-end step time. With expert parallelism alone, correlated placement and deduplication reduce A2A time by $1.16\times$–$1.38\times$ under top-2 routing and by $1.32\times$–$1.95\times$ under top-6 routing. With token shuffling added, A2A time is reduced by $1.40\times$–$1.74\times$ under top-2 and by $1.46\times$–$2.63\times$ under top-6.

2 MOTIVATION

2.1 ALL-TO-ALL COST IN MOE PRETRAINING

We begin by measuring how much of an MoE training step is spent in the dispatch and combine collectives. Each MoE layer runs both collectives in the forward pass and again in the backward pass. The cost of these collectives depends on two configuration choices. A larger expert-parallel degree spreads experts over more GPUs, so the fraction of a token's experts on its own GPU falls as $1/\mathrm{EP}$. Beyond a single node, however, the exchange also moves from the intra-node fabric to the inter-node network. A larger top-k raises the number of activation rows each token contributes to both collectives, so top-6 moves three times the volume of top-2.

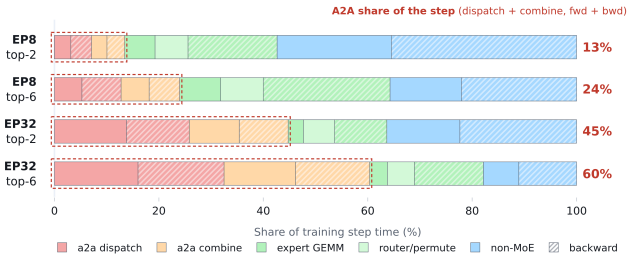
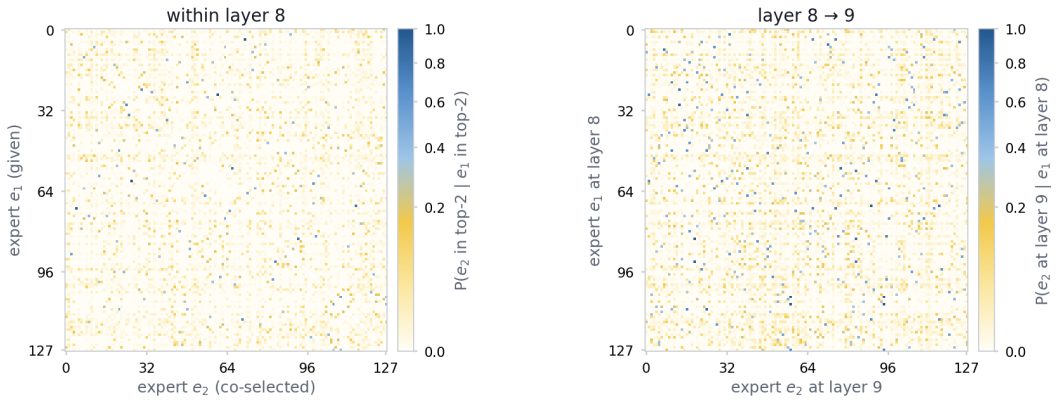


Figure 1: Share of training step time by component at EP8 and EP32 with top-2 and top-6 routing, using the Megatron-LM all-to-all dispatcher and contiguous placement. Dashed boxes mark the all-to-all share.

We measure a 12-layer MoE model (d_{model}=4096, with 128 experts of FFN width 1024 per layer(see Section 4 for the detailed configuration). Figure 1 shows that the two effects compound. We measure all-to-all time as the wall-clock time the step spends inside the dispatch and combine calls, which includes time spent waiting for other GPUs. Widening the EP group from a single node (EP8) to four nodes (EP32) raises the all-to-all share of step time from 13.5 % to 44.8 % under top-2 and from 24.0 % to 60.4 % under top-6. At fixed EP8, moving from top-2 to top-6 raises it from 13.5 % to 24.0 %. At EP32 with top-6, the two collectives take more of the step than expert computation and all non-MoE computation combined.

2.2 EXPERT CORRELATION WITHIN AND ACROSS LAYERS

Crucially, the measurements above use contiguous placement which ignores the correlations between the experts a token selects. Once experts have specialized, these selections follow two regular patterns, which we exploit in this paper. We measure both on the top-2 and top-6 models from Section 2.1. All statistics in this section are computed offline. We record the routing decisions of the trained models on held-out iterations and apply each placement and predictor to the recorded routes without running the dispatcher.



(a) Within layer 8: $P(e_2 \in \text{top-2} \mid e_1 \in \text{top-2})$.

(b) Across layers $8 \to 9$: $P(e_2 \text{ at } 9 \mid e_1 \text{ at } 8)$.

Figure 2: Expert correlation in the top-2 model, measured on 524k tokens from iterations 1983–1995. Blue marks conditional probabilities above 0.3.

Within a Layer, Experts Are Selected in Stable Pairs. Figure 2a(a) shows, for layer 8, the probability that e_2 is a token's second expert given that e_1 is its first. The dark cells are pairs almost always chosen together: 14 pairs are chosen with at least 50 % confidence and carry 16 % of all tokens, and 64 of the 8,128 possible pairs carry 42 %, where independent routing with the same loads would give 1.6 %. For 73 of the 128 experts one partner appears in at least 30 % of its tokens.

		within layer			across layers: local experts	
top-k, EP	placement	same GPU	GPUs	deduplication	token moved	token stays
top-2, EP8	contig.	11.9	1.88	5.9	33.4	12.5
	corr.	52.9	1.47	26.4	57.7	12.5
top-2, EP16	contig.	5.5	1.95	2.7	27.6	6.2
	corr.	45.3	1.55	22.7	51.0	6.2
top-2, EP32	contig.	2.2	1.98	1.1	24.5	3.1
	corr.	35.8	1.64	17.9	42.4	3.1
top-6, EP8	contig.	0.0	4.49	25.2	25.1	12.5
	corr.	18.0	2.54	57.7	60.4	12.5
top-6, EP16	contig.	0.0	5.24	12.7	18.3	6.2
	corr.	8.3	3.08	48.6	49.2	6.2
top-6, EP32	contig.	0.0	5.67	5.5	14.1	3.1
	corr.	0.0	3.89	35.1	32.6	3.1

Table 1: Offline routing statistics on held-out iterations 1983 and 1987 (fit on 1991–1999); all values in % except *GPUs*. *Same GPU*: tokens whose top-k experts share a GPU. *GPUs*: distinct destinations per token. *Deduplication*: rows removed as duplicates. *Local experts*: share of a token's next-layer experts on its own GPU when it is moved to the GPU predicted from its current experts (T/EP tokens per GPU) and when it stays on its home GPU ($1/\mathrm{EP}$).

A token is sent once per GPU its experts occupy, so co-locating a frequent pair halves its traffic under a deduplicating dispatcher. Contiguous placement (expert e on GPU $\lfloor e/(E/\mathrm{EP})\rfloor$) does this only by chance; Correlated placement does it on purpose. Table 1 (within layer) counts rows, one row being one copy of a token's hidden state sent to one GPU. At EP8, correlated placement raises the share of top-2 tokens whose experts share a GPU from 12 % to 53 %, and the rows removed by deduplication from 6 % to 26 % under top-2 and 25 % to 58 % under top-6. At EP32 a GPU holds only four experts, so a top-6 token's experts can never all share one, yet rows removed still rise from 5.5 % to 35.1 %.

Across Layers, the Current Experts Predict the Next Ones. Figure 2a(b) plots, for each expert e_1 at layer 8, the probability that a token routed to e_1 selects expert e_2 at layer 9. Most layer-8 experts send a large share of their tokens to a single layer-9 expert: 112 of the 128 send at least 30 % to one specific expert, and the median share is 48 %.

This correlation can be used to move *tokens*. Before the router of layer $i{+}1$ runs, the experts chosen at layer i already indicate which GPU will hold most of the token's next experts, and under sequence parallelism (Section A.1.2) the token can be moved there during the attention block. Every GPU must own exactly T/EP tokens, so not every token gets its predicted GPU. Table 1 (across layers) reports, under this constraint, the share of a token's next-layer experts that are on its own GPU. A token that stays on its home GPU has $1/\mathrm{EP}$ of them local. A token moved to its predicted GPU has 58 %, 51 % and 42 % at EP8, EP16 and EP32 for top-2 and 60 %, 49 % and 33 % for top-6 under correlated placement. Placement matters here too: with contiguous placement the moved token has only 33 % at EP8, because its next experts are spread over more GPUs.

3 Implementation

3.1 Within-Layer Correlation: Correlated Expert Placement

As discussed in Section 2.2 and shown in Figure 2a, some experts within a layer are frequently selected together by the same tokens. We use this structure to reduce all-to-all (A2A) communication with two techniques. *Correlated expert placement* puts frequently co-selected experts on the same GPU, and a *deduplicating dispatcher* sends each token to each EP rank at most once.

Figure 3 shows the A2A communication under the baseline, with deduplication, and with both deduplication and correlated expert placement. $\rightarrow$ denotes rows sent to a remote rank, $\rightarrow$ denotes rows whose expert is on the token's own rank, and $\rightarrow$ denotes duplicate rows that deduplication removes. (1) shows contiguous placement with the conventional dispatcher, where each (token, expert) assignment produces a separate A2A row. When a token selects several experts on the same destination rank, its hidden state is sent there several times. (2) adds deduplication. Assignments that share the same (token, destination rank) are merged, so the hidden state is sent to that rank once. (3) adds correlated expert placement, which puts frequently co-selected experts on the same rank. This increases

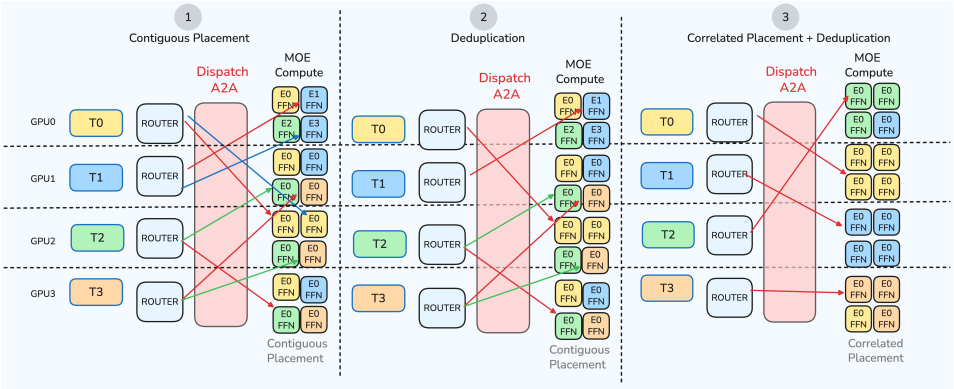


Figure 3: Reducing A2A with within-layer expert correlation.

token–expert locality and gives deduplication more rows to merge. Neither technique changes the router's expert assignments or the expert computation.

Deduplicating Dispatcher. Conventional top-k dispatch sends one copy of the token's hidden state per $(\text{token}, \text{expert})$ assignment, even when several selected experts reside on the same EP rank. We instead send one row per unique $(\text{token}, \text{destination rank})$ pair. Each row carries the IDs and routing probabilities of the token's experts on that rank. The receiving rank uses them to give each of its selected experts a copy of the token and to group the copies by expert for the expert computation, so no separate collective is needed for this metadata. The combine path works the same way. The expert rank weights the outputs of the token's experts by their routing probabilities, sums them, and returns one partial output per $(\text{token}, \text{rank})$ pair. Dispatch and combine therefore send one row per token for each rank that holds at least one of its experts. Deduplication is an existing technique (Zhao et al., 2025), and we implement it in Megatron-LM as the basis for our correlated placement algorithms.

Correlated Placement. For each MoE layer, we build a correlation graph W, where $W[e, e']$ counts the tokens that select both experts e and e'. We look for a placement that puts the same number of experts on every GPU and maximizes the total weight $W[e, e']$ of the expert pairs that share a GPU. Algorithm 1 in the appendix performs a Kernighan–Lin-style local search (Kernighan & Lin, 1970). It starts from contiguous placement and swaps pairs of experts between GPUs whenever a swap increases this total. Every GPU holds exactly E/EP experts, so the placement does not change the memory used for expert parameters.

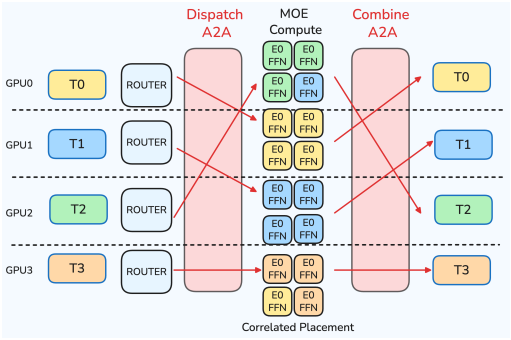


Figure 4: TP–EP baseline with correlated expert placement. Each token stays on the rank of its canonical sequence slice, so its rows still cross the network even when its experts share one GPU.

For EP groups that span multiple nodes, we partition in two steps. We first partition the experts across nodes, which keeps co-selected experts on the same node and reduces inter-node traffic. We then partition each node's experts across its GPUs, which keeps co-selected experts on the same GPU where possible. The resulting placement is a per-layer permutation of global expert IDs and does not change router decisions or expert parameters.

Effect of Placement and Deduplication. The two techniques are complementary. Deduplication can exploit expert co-location that occurs even under contiguous placement, while correlation-aware placement deliberately increases the frequency of such co-location. With EP8, deduplication under contiguous placement removes 6 % of rows for top-2 and 25 % for top-6 routing. Combining

correlation-aware placement with deduplication increases these reductions to 26 % and 58 %, respectively. At EP16 across two nodes, the corresponding reductions are 23 % and 49 % (Table 1). The resulting reductions in A2A communication and end-to-end step time are evaluated in Section 4.

3.2 Cross-Layer Correlation: Token Shuffling

Expert placement decides where experts live and does not change where tokens live. With TP–EP parallelism, the MoE layer runs on the sequence shard that the reduce-scatter after attention assigns to each rank. Each rank thus owns a fixed, contiguous slice of the sequence, and a token's rank is set by its position alone. Even when the experts a token needs have been placed together on one GPU, the token is actually on that GPU with probability only $1/\mathrm{EP}$. Thus, with probability $1 - 1/\mathrm{EP}$ the token's row must still cross the wire in the dispatch and returns in the combine. Figure 4 shows this for four ranks under correlated placement. The conventional reduce-scatter places T0 on GPU0, T1 on GPU1 and so on, although T0's experts sit together on GPU1 and T1's on GPU3, so their rows still leave the owner's GPU.

The cross-layer correlation of Section 2.2 (Figure 2b) lets us move the token as well. The experts a token used at layer ℓ predict, well above chance, which GPU will host its experts at layer $\ell{+}1$, and this prediction is available a whole attention block before the layer-$(\ell{+}1)$ router runs. Token shuffling (Figure 5) uses this window. We call the GPU that holds a token in the MoE layer its *owner*. The reduce-scatter that follows attention first permutes tokens so that each goes to its predicted owner and then scatters them, so the following MoE dispatch finds that most of the token's experts are now local. The rest of this section describes where ownership can change without extra communication, how the owner is predicted, and how the planner is hidden behind attention.

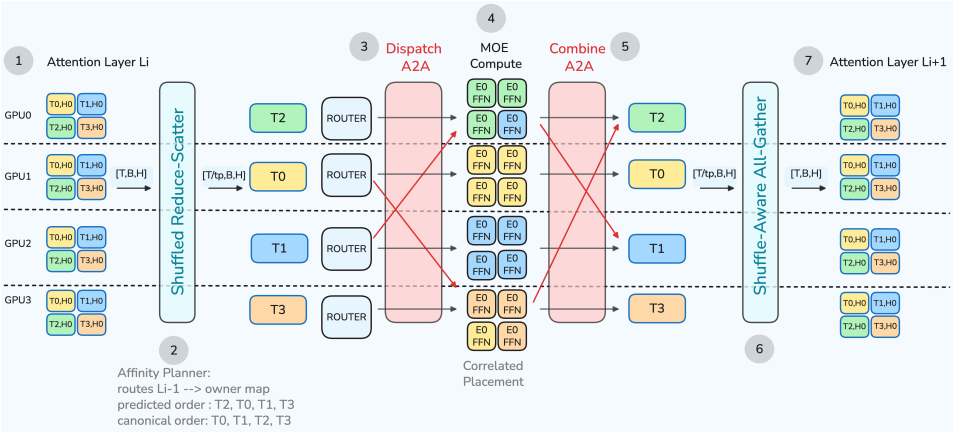


Figure 5: Token shuffling end to end. The affinity planner (2) predicts each token's owner rank from its layer-ℓ routes. The shuffled reduce-scatter (3) delivers each row to that rank, so the MoE dispatch (4) (5) finds most experts local. The shuffle-aware all-gather (6) restores canonical order before the next attention block (7).

Changing Ownership Inside Existing Collectives. The construction requires the TP group and the EP group to be the same set of ranks ($\mathrm{TP} = \mathrm{EP}$). In TP–EP, a reduce-scatter follows the attention output projection and returns the sequence to its sharded layout. An all-gather precedes the next block's QKV projection and reassembles the full sequence on every rank (Figure 5). Both collectives already move every token between ranks. Which rank receives which token in the reduce-scatter is set by indexing, so changing it does not change the bytes sent. We therefore fuse a permutation into each collective. The fused reduce-scatter sends every token to its planned owner instead of the rank that holds it in the default contiguous layout. It also moves the token's row of the residual stream, so the residual addition happens on the new owner. The fused all-gather restores the original token order on the way back, because attention depends on token order. The MoE layer sees a local slice whose tokens mostly have their experts on the same GPU, and the attention layer is unchanged. Token shuffling thus adds no communication. Its two fused permutations cost a few milliseconds per step, compared with the hundreds of milliseconds they remove from the all-to-all (Section 4).

Predicting the Owner. The shuffle planner answers one question per token: given the experts it just visited, on which rank will its next experts most likely live? Routing is strongly correlated across layers, so the pair of experts a token used at the two preceding MoE layers is a good predictor of where it lands next. Offline, from routing traces and the chosen placement, we count for every MoE layer how often each such pair sends the token to each rank; when a pair has too few observations we fall back to the single preceding expert. At run time the previous layers' routes are the only input, and the planner adds no collective. The lookup gives each token a preferred owner and a confidence. Sequence parallelism requires every rank to own exactly T/EP tokens, so tokens are assigned in decreasing confidence, each to its most preferred rank with capacity left. We give the full algorithm in Algorithm 2.

Hiding the shuffle planner. Prediction and quota assignment cost about a third of the forward all-to-all time at TP16 EP16, so if executed sequentially, this would forfeit most of the gain. However, crucially, the plan depends only on the layer-i routes and is needed only at the layer-$(i{+}1)$ reduce-scatter, so we can run the planner on a side stream under the experts and the next attention block and join it with a single event wait (Figure 6). The cost in practice is thus only a few milliseconds per step plus one fused permute-plus-residual kernel.

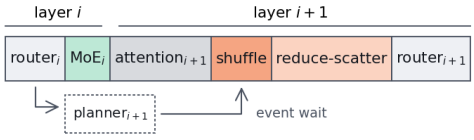


Figure 6: The planner runs on a side stream between the layer-i routes and the layer-$(i{+}1)$ reduce-scatter.

4 Experimental Results

All experiments use the two models of Section 2.1. Both are 12-layer MoE models with $d_{\mathrm{model}} = 4096$ and 128 experts of FFN width 1024 per layer, one with top-2 and one with top-6 routing. Both use a sequence-level auxiliary loss with coefficient 10^{-2} for load balancing. We train each for 2.1 billion tokens (2000 steps) on FineWeb-Edu (Penedo et al., 2024) with a sequence length of 8192 and a global batch size of 128. All evaluations load the step-2000 checkpoint of the respective model and run on held-out data from the same corpus, so the routing decisions are those of a trained model. We take all measurements on nodes of eight AMD Instinct MI300X GPUs, fully connected within a node by xGMI (one direct link between every pair of GPUs) and connected across nodes by eight 100 Gb/s RoCE NICs per node, one per GPU. Each experiment is measured twice. The first run of 14 clean steps gives the median step time we report. The second run of 14 steps enables the region profiler and all-to-all metrics, which provide the time and traffic breakdowns. Speed-ups are relative to the baseline, which is the Megatron-LM all-to-all dispatcher with contiguous placement. We compare three configurations against it. These are deduplication, deduplication with correlated expert placement, and, on layouts with $\mathrm{TP} = \mathrm{EP}$, token shuffling on top of both. The figures label the baseline and the three configurations B, D, P and S. We implement all methods in Megatron-LM (Shoeybi et al., 2019). We note that the expert correlation does not change with model size or routing, so we expect the gains reported here to carry over to larger models.

4.1 Building Expert Correlation Within and Across Layers

Correlated placement and the token-shuffling predictor both use correlation tables fit offline from recorded routing. We measure how many tokens these tables need to converge and how quickly they go stale during training. We test both cases on held-out data.

A Few Thousand Tokens Suffice. To measure how much routing data the tables need, we fit them on different numbers of tokens from iterations 1983–1995 and evaluate them on held-out iteration 1999. With tables fit on 1,024 tokens, correlated placement removes 23.6 % of dispatched rows at top-2 EP8 and 57.9 % at top-6. With 4,096 tokens, the Pearson correlation between the fitted and the held-out co-selection matrices is 0.93 for top-2 and 0.98 for top-6, and the placement and prediction results are within one percentage point of those obtained with all 524k tokens. A single microbatch from one rank is therefore enough to fit both tables. The tables capture a property of the trained router and do not depend on the particular tokens used to fit them and that the correlation structure is robustly discoverable from relatively few tokens.

The Structure Forms Early and Then Holds. To measure how early the correlations form, we fit the tables on routing from a single training step and evaluate them on routing from step 1999, the end of training. Tables fit at step 199 already give about half of the final benefit. At top-2 EP8, the placement built from them removes 14.4 % of dispatched rows, compared with 26.5 % for tables fit at the end of training. At top-6 the corresponding numbers are 42.7 % and 57.7 %. Tables fit at step 1000 come within 2–3 percentage points of the final values, and tables fit at step 1400 or later give nearly the same results as the final tables. The correlation between the fitted and the final co-selection matrices follows the same pattern, rising from 0.4–0.5 at step 199 to above 0.93 from step 1400 on. The predictor improves at the same rate. At top-2 EP8, its hit rate is 12.7 % for tables fit before routing has formed, close to the chance level of 12.5 %. It rises to 54 % for tables fit at step 1000 and 57 % for tables fit at step 1800. For our 2000-step runs, fitting the tables once in the second half of training is therefore enough. For longer runs, we expect refitting about every 1000 steps to be sufficient, and that each refit needs only one microbatch of recorded routes.

4.2 Correlated Expert Placement

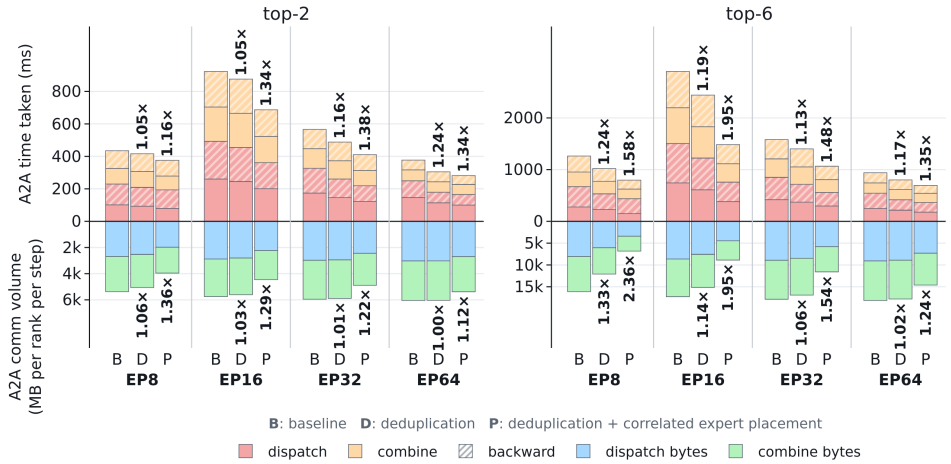


Figure 7: All-to-All Cost Under Expert Parallelism. A2A time (top) and communication volume per rank (bottom) at TP1 with EP8–EP64, for top-2 and top-6 routing. B is the baseline, D adds deduplication, and P adds correlated placement to deduplication.

All-to-All Communication Volume Correlated expert placement accounts for most of the reduction in all-to-all traffic. Under contiguous placement, deduplication removes only the duplicates that land on the same rank by chance. This is 6 % of rows at top-2 and 25 % at top-6 for EP8, and 3 % and 12 % at EP16. With correlated placement, the communication volume falls by 1.36× at top-2 EP8 and 2.36× at top-6 EP8 relative to the baseline. The reduction grows with top-k, since more of a token's experts can share a rank. The effect holds as EP widens and each rank holds fewer experts, at 1.29×/1.22×/1.12× for top-2 and 1.95×/1.54×/1.24× for top-6 at EP16/32/64. Dispatch and combine carry identical byte counts by construction, since every row that goes out comes back, so the byte reduction applies identically to the backward pass also.

All-to-All Communication Time Across nodes, A2A time falls by 1.34×/1.38×/1.34× for top-2 and 1.95×/1.48×/1.35× for top-6 at EP16/32/64. Within a node (EP8), where the collective runs over xGMI, it falls by 1.16× at top-2 and 1.58× at top-6. The reduction appears in all four collectives, dispatch and combine in both the forward and the backward pass.

4.3 Token Shuffling

Token shuffling applies where the TP group and the EP group coincide. We evaluate it at TP8 EP8 on one node and TP16 EP16 across two nodes. Figure 8 shows its all-to-all results alongside deduplication and correlated placement, and Figure 10 in the appendix shows the end-to-end step time.

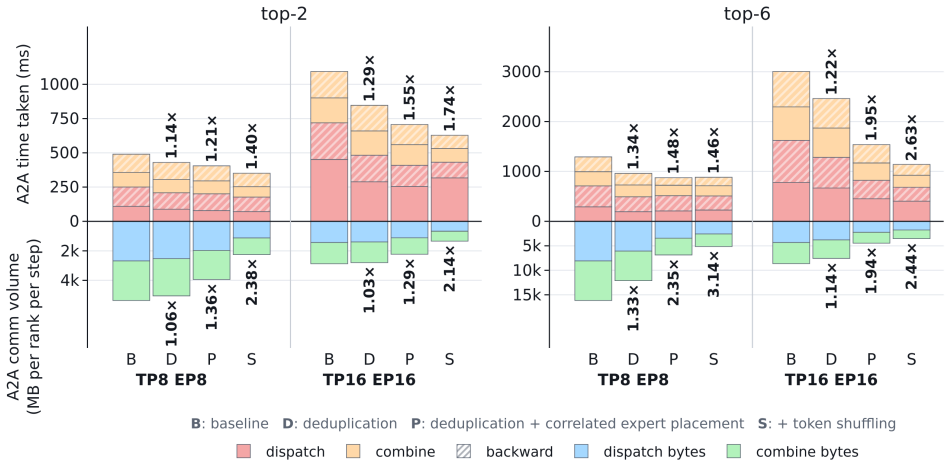


Figure 8: All-to-All Cost on the TP = EP Layouts. Same encoding as Figure 7, with token shuffling as the fourth configuration (S). The shuffle places tokens on the GPUs that hold their experts, so more of the dispatch stays on the device and less traffic crosses the network.

Token–Expert Locality Without token shuffling, a token's expert slots are on its own GPU with probability $1/\mathrm{EP}$, which is 12.5 % at EP8 and 6.3 % at EP16. Token shuffling raises this to 59 % and 53 % for top-2 and to 57 % and 46 % for top-6, close to the offline hit rates in Table 1. The dispatcher still counts these rows, but they stay on the GPU and never cross the network. Relative to the baseline, wire bytes fall by 2.4× at EP8 and 2.1× at EP16 for top-2, and by 3.1× and 2.4× for top-6. With placement alone, the reductions are 1.4× and 1.3× for top-2 and 2.4× and 1.9× for top-6.

All-to-All Communication Time At TP16 EP16, token shuffling raises the A2A reduction from 1.55× (placement alone) to 1.74× at top-2 and from 1.95× to 2.63× at top-6. It raises the step speedup at top-6 from 1.28× to 1.34×. At top-2 the step speedup is 1.03×, because the all-to-all is only 21 % of that step. On a single node (TP8 EP8), token shuffling reduces bytes by a similar or larger factor, but time much less. The A2A reductions are 1.40× at top-2 and 1.46× at top-6, and the step speedups are 1.01× and 1.08×. At top-6 on one node, the A2A reduction with token shuffling is slightly below that of placement alone (1.48×). On xGMI the collective time is set by the busiest GPU pair and does not follow the aggregate bytes. Concentrating a token's rows on one owner raises the traffic on the busiest link even as it lowers the total. The two fused permutations of token shuffling add a few milliseconds per step. The planner runs behind attention, and the main stream waits less than a millisecond per step for it (Section 3.2).

5 Conclusion

We showed that expert routing in MoE pretraining is correlated both within a layer and across layers, and that these correlations form within about one billion tokens and then stay stable. Correlated expert placement uses the within-layer correlation to put co-selected experts on the same GPU, and a deduplicating dispatcher sends each token to each GPU once. Token shuffling uses the cross-layer correlation to move each token to the GPU predicted to hold its next experts, inside the reduce-scatter that sequence parallelism already performs. Neither method changes routing decisions or expert parameters. On nodes of eight MI300X GPUs, the two methods reduce all-to-all time by 1.16–2.63× and end-to-end step time by up to 1.41×, with the largest gains at top-6 when the EP group spans several nodes. Our evaluation covers one 12-layer model with top-2 and top-6 routing, trained for 2.1B tokens. We believe the expert correlation holds across model families and scales, and leave other parallelism layouts, such as context parallelism, and EP inference to future work.

REFERENCES

Yutian Chen, Cong Li, Yucheng Wang, and Ming Wei. MoonEP: A perfectly balanced expert parallelism library via dynamic redundant experts. `https://github.com/MoonshotAI/MoonEP`, 2026.

Damai Dai, Chengqi Deng, Chenggang Zhao, Runxin Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Dennis Liu, Zijie Yan, Xin Yao, Tong Liu, Vijay Korthikanti, Evan Wu, Shiqing Fan, Gao Deng, Hongxiao Bai, Jianbin Chang, et al. MoE parallel folding: Heterogeneous parallelism mappings for efficient large-scale MoE model training with Megatron Core. *arXiv preprint arXiv:2504.14960*, 2025.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 37, pp. 30811–30849, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatele. A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training. In *Proceedings of the 37th International Conference on Supercomputing*, pp. 203–214, 2023.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. OpenMoE: An early effort on open mixture-of-experts language models. In *International Conference on Machine Learning*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jinghan Yao, Quentin Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K DK Panda. Exploiting inter-layer expert affinity for accelerating mixture-of-experts model inference. In *2024 IEEE International parallel and distributed processing symposium (IPDPS)*, pp. 915–925. IEEE, 2024.

Mingshu Zhai, Jiaao He, Zixuan Ma, Zan Zong, Runqing Zhang, and Jidong Zhai. {SmartMoE}: Efficiently training {Sparsely-Activated} models through combining offline and online parallelization. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pp. 961–975, 2023.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. DeepEP: an efficient expert-parallel communication library. `https://github.com/deepseek-ai/DeepEP`, 2025.

A Appendix

A.1 Parallelism Background

A.1.1 Expert Parallelism

When the experts of a layer no longer fit in the memory of one GPU, expert parallelism (EP) partitions them across a group of GPUs, typically alongside data parallelism. With $E{=}128$ experts and $\mathrm{EP}{=}8$, for example, each GPU hosts $E/\mathrm{EP}{=}16$ experts. A token routed to an expert on another GPU is sent there by the dispatch all-to-all and returned by the combine all-to-all, in both the forward and the backward pass. The first panel of Figure 3 shows this pattern. Every token–expert assignment that crosses a GPU boundary adds a row to both collectives.

A.1.2 Hybrid Parallelism: Tensor–Expert Parallelism

For large models and long-context training, expert parallelism alone is not enough to fit the model in GPU memory, and additional model parallelism is needed. Tensor parallelism (TP) (Shoeybi et al., 2019) shards the computation of each layer across GPUs and is used in both pretraining and inference.

In a common hybrid TP–EP configuration, attention and the MoE layer use different token layouts. Attention is partitioned across the TP ranks, and the MoE layer operates on sequence shards distributed across the same ranks (Korthikanti et al., 2023; Liu et al., 2025; Singh et al., 2023). Two collectives move between these layouts. After attention, a reduce-scatter sums the partial TP outputs and partitions the sequence, which leaves each rank with a contiguous subset of tokens. These tokens then pass through the dispatch all-to-all, the expert computation and the combine all-to-all. After the MoE layer, an all-gather reconstructs the full sequence on every TP rank for the next Transformer block.

Token shuffling replaces this reduce-scatter and all-gather with shuffled versions (Figure 5), and it requires the TP group and the EP group to be the same GPUs ($\mathrm{TP} = \mathrm{EP}$). The shuffled reduce-scatter sends each token to the rank predicted to hold most of its experts in the coming MoE layer, based on the experts it selected in the previous layers. The shuffle-aware all-gather restores the canonical token order before the next attention layer. More of each token's experts are then on its own GPU, and the dispatch and combine all-to-alls send fewer bytes across the network. Section 3.2 describes the mechanism in detail. Experts are not tensor-parallel in this work. Some latency-sensitive inference deployments shard experts with tensor parallelism, which we do not consider.

A.2 Algorithms and Additional Results

Algorithm 1 Correlated Placement

Input: routing sets $\{\mathcal{S}_\ell(t)\}$, experts E, EP ranks P, nodes N, budget $I_{\max}$
Output: balanced placement π_ℓ per MoE layer
for each MoE layer ℓ **do**
  $W[e, e'] \leftarrow \sum_t \mathbf{1}[e, e' \in \mathcal{S}_\ell(t)]$
  if $N = 1$ **then**
    $\pi_\ell \leftarrow \textsc{Partition}(W, P, I_{\max})$
  else
    $\nu \leftarrow \textsc{Partition}(W, N, I_{\max})$
    for node $n = 0, \ldots, N-1$ **do**
      $\mathcal{M}_n \leftarrow \{e : \nu(e) = n\}$
      $\rho_n \leftarrow \textsc{Partition}(W_{\mathcal{M}_n}, \frac{P}{N}, I_{\max})$
      $\pi_\ell(e) \leftarrow n\frac{P}{N} + \rho_n(e) \;\; \forall e \in \mathcal{M}_n$
    end for
  end if
end for

procedure $\textsc{Partition}(W, G, I_{\max})$ ▷ balanced
  Initialize G contiguous groups of size E/G
  for $i = 1, \ldots, I_{\max}$ **do**
    $(a, b) \leftarrow$ best swap by intra-group W gain
    if no improving swap exists **then break**
    Swap a and b
  end for
  return expert-to-group assignment
end procedure

Algorithm 2 Affinity-guided token ownership for shuffled TP ($\mathrm{TP} = \mathrm{EP} = P$)

Offline, from routing traces and the placement π: for each MoE position m, tables $F_m[e, r]$, $S_m[e', e, r]$, supports $n_m[e', e]$ (and optionally $Q_m[\mathrm{tok}, r]$); back-off β, cap $w_{\max}$.
Run time, at the boundary before MoE position m; input: top-k sets $\mathcal{S}_{m-1}(t)$, $\mathcal{S}_{m-2}(t)$ of every token t in the global sequence, gathered with the attention all-gather.
// Side stream, overlapped with the attention block of the layer that hosts MoE position m
for each token t **do** ▷ predict a distribution over owner ranks
  $d_t \leftarrow \frac{1}{k^2} \sum_{e \in \mathcal{S}_{m-1}(t)} \sum_{e' \in \mathcal{S}_{m-2}(t)} \Big[(1 - w)\, F_m[e, \cdot] + w\, S_m[e', e, \cdot]\Big]$, $w = \min\Big(\frac{n_m[e', e]}{n_m[e', e] + \beta},\, w_{\max}\Big)$
  optionally $d_t \leftarrow (1 - w_t)\, d_t + w_t\, Q_m[\mathrm{tok}(t), \cdot]$ with the same back-off on the token's support
  $d_t \leftarrow d_t / \sum_r d_t[r]$
end for
$\mathrm{margin}(t) \leftarrow d_t^{(1)} - d_t^{(2)}$ ▷ best minus second-best rank probability
$\mathrm{cap}[r] \leftarrow T/P$ for all r
for each token t in decreasing order of $\mathrm{margin}(t)$ **do** ▷ balanced assignment
  $\mathrm{owner}(t) \leftarrow$ the rank r with the largest $d_t[r]$ among those with $\mathrm{cap}[r] > 0$; $\mathrm{cap}[r] \leftarrow \mathrm{cap}[r] - 1$
end for
Build order (canonical token id at every position of the new layout), its inverse, and the residual-row indices of the rows each rank owns now and will own next.
// Main stream, at the attention output projection of the same layer
Wait on the planner's completion event.
Reduce-scatter the projected rows to $\mathrm{owner}(\cdot)$ instead of the canonical slice; permute and add the residual in the same fused kernel.
The MoE layer at position m dispatches from the owner layout; the layout persists until the next plan.

A.3 End-to-End Results

Non-MoE computation takes the same time in all configurations (blue in Figure 9). The step speedup therefore depends on how much A2A time is saved and on how large a share of the step the A2A takes. At top-2 the A2A share is 13–52 %, and correlated placement speeds up the step by 1.05× at EP8, 1.13× at EP16, 1.14× at EP32 and 1.11× at EP64. At top-6 the share is 24–60 %, and the speedups are 1.14× at EP8, 1.41× at EP16, 1.25× at EP32 and 1.23× at EP64. The speedup is larger at top-6 and is largest when the A2A crosses node boundaries.

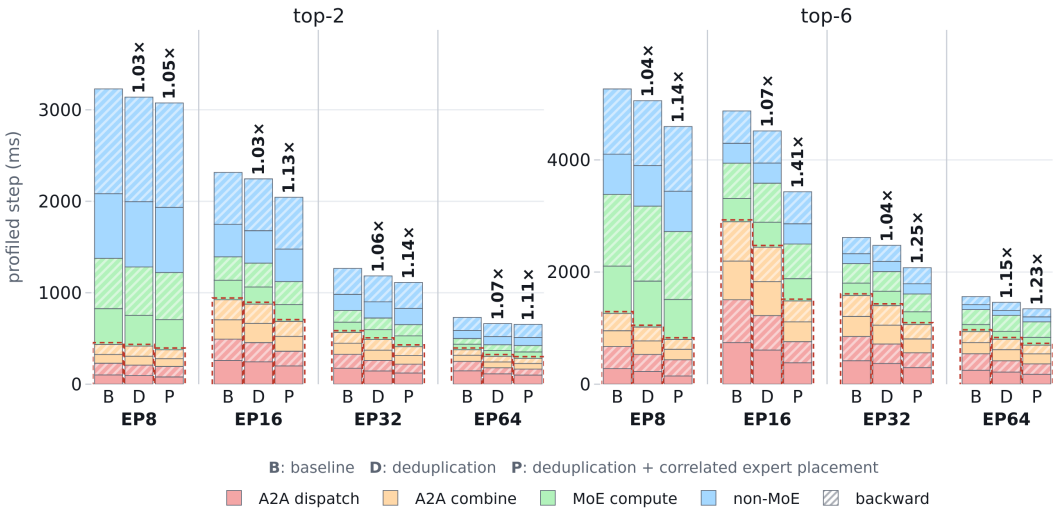


Figure 9: Step-time breakdown under expert parallelism, top-2 and top-6 at TP1, EP8–EP64.dashed boxes mark the A2A share of each bar.

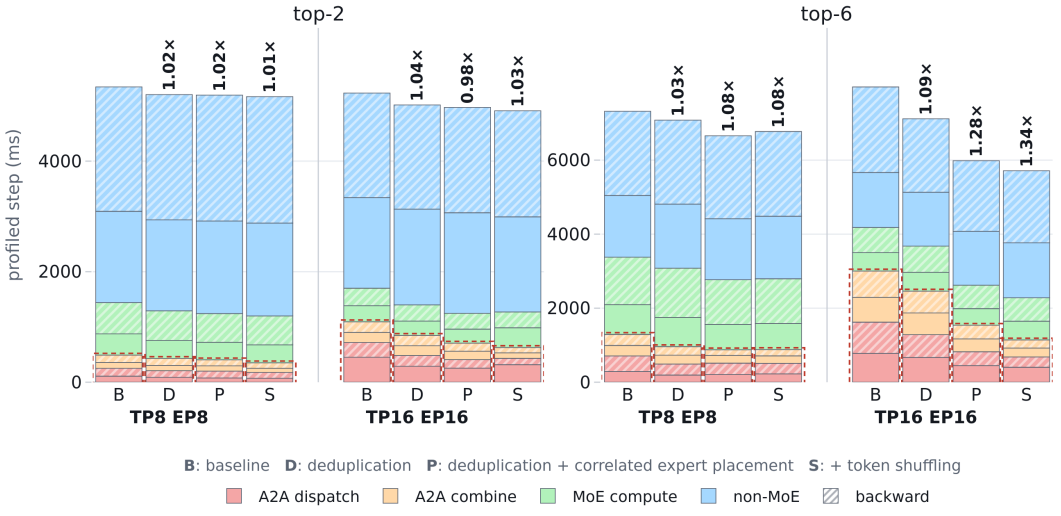


Figure 10: Step-time breakdown on the TP = EP layouts (TP8 EP8, TP16 EP16) with token shuffling added as S. Same encoding as Figure 9

On the TP = EP layouts (Figure 10) the tensor-parallel collectives inflate the non-MoE part, so the A2A is a smaller share of the step: 9–21 % at top-2 and 18–38 % at top-6. Token shuffling cuts the A2A time itself by 1.40× and 1.74× at top-2 and by 1.46× and 2.63× at top-6 (TP8 EP8 and TP16 EP16), but the step speedup follows the share. At top-2 it stays within 1.01–1.04× for every

variant, and the 0.98× of correlated placement at TP16 EP16 comes from a 5 % longer non-MoE part, not from the A2A. At top-6 the gains are 1.08× at TP8 EP8 and 1.34× at TP16 EP16, where shuffling adds 6 points on top of correlated placement.

A.4 Ablation Over Routing Strategies

Gains Follow Each Router's Co-Selection Structure. Our methods depend on the router producing correlated expert choices. To test how the load-balancing method affects this, we trained two more top-6 models that differ from the main top-6 model only in the balancing method. The first uses a sequence-level auxiliary loss ten times stronger than the main model's (coefficient 10^{-1} instead of 10^{-2}). It gives the most even expert load of the three, with a maximum-to-mean load ratio of 1.20. The second uses the auxiliary-loss-free method of DeepSeek-V3 (Liu et al., 2024), which adds a per-expert bias to a sigmoid router. Its load ratio is 1.37, and it reaches the lowest loss of the three (4.39, against 4.46 for the main model and 4.64 for the stronger auxiliary loss). We evaluate the same configurations on all three models.

Gains Follow Each Router's Co-Selection Structure. With correlated placement, deduplication removes 48.8 % of dispatched rows for the main model, 40.5 % for the stronger auxiliary loss and 35.9 % for the auxiliary-loss-free router; the predictor hit rates are 49.5 %, 41.1 % and 36.6 %, respectively. The step speedups preserve this ordering on every layout (Figure 11), reaching 1.41×, 1.30× and 1.26× at TP1 EP16 and 1.28×, 1.24× and 1.20× at TP16 EP16. The auxiliary-loss-free router thus leaves the least exploitable structure, even though its load is less balanced than that of the stronger auxiliary loss: a router can balance load well and still select experts in correlated groups. Across all three routers, deduplication with correlated placement yields at least a 1.20× speedup on every EP16 layout and at least 1.26× at TP1 EP16. Balancing methods that spread co-selected experts apart leave less structure for correlated placement to exploit.

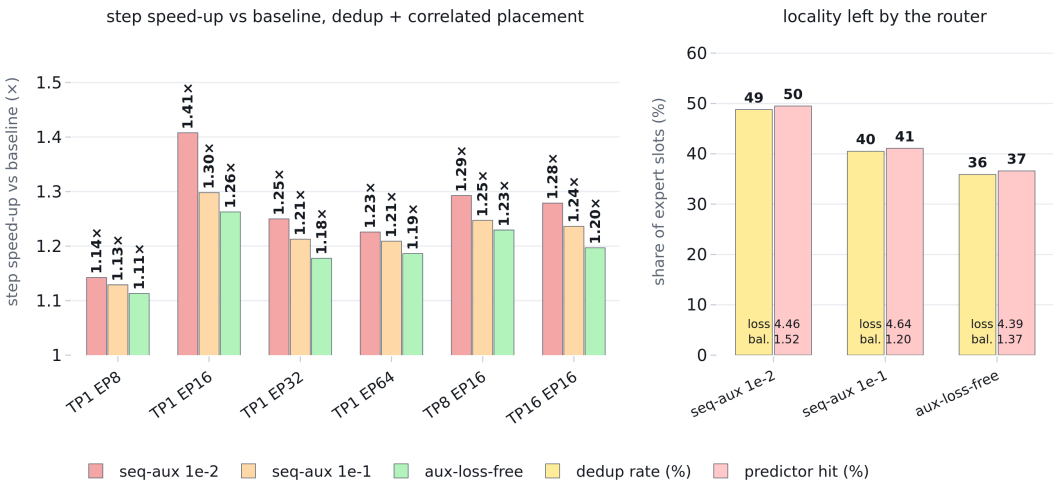


Figure 11: Router ablation on three top-6 models that differ only in load balancing: the main recipe (sequence auxiliary loss 10^{-2}), a 10× stronger auxiliary loss (10^{-1}), and the DeepSeek-V3 auxiliary-loss-free router. Left: step speed-up of deduplication + correlated placement over that model's baseline, per layout. Right: the structure each router leaves to exploit, as the dedup rate under correlated placement and the predictor hit rate, with each model's loss and load balance (max/mean expert load) printed on the bars.